§1 ЛІНІЙНІ ПРОГРАМИ

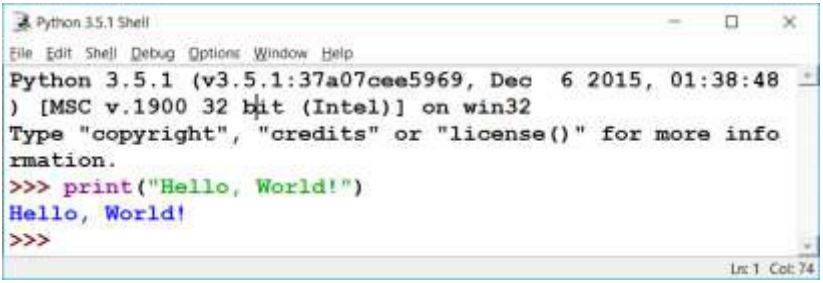
Перша програма на Python

Традиційно, першою програмою при вивченні будь-якої мови програмування є програма «Hello, World!». Ця програма виводить на екран (або в консоль) привітання «Hello, World!». Щоб написати її у Python, досить однієї інструкції

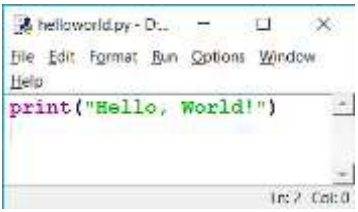
```
print("Hello, World!")
```

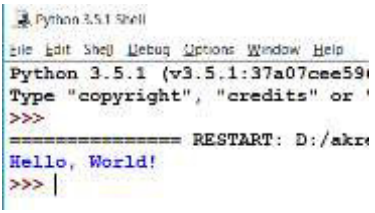
Виконати цю програму можна двома способами

- Ввести у консолі Python код програми (наприклад у інтерактивному режимі IDLE після символів очікування >>>) і натиснути клавішу Enter:



- Зберегти код програми у (текстовому) файлі з розширенням .py та запустити його на виконання. Наприклад, у IDLE для створення файлу необхідно виконати ланцюг File → New File з меню (або натиснути Ctrl + N). Для запуску програми необхідно виконати ланцюг Run → Run Module з меню (або натиснути клавішу F5)





Синтаксис

Синтаксис мови програмування це набір правил, що описує комбінації символів алфавіту, які вважаються правильно структурованою програмою або її фрагментом.

Отже, синтаксис визначає як буде виглядати програма на цій мові, зокрема, як пишуться оператори, оголошення і інші мовні конструкції.

Основні правила

Основні правила синтаксису мови програмування Python полягають у такому:

- Кінець рядка є кінцем інструкції;
- Допускається записувати кілька інструкцій у одному рядку. При цьому їх розділяють крапкою з комою

```
a = 1; b = 2; print(a, b)
```

- Допускається записувати одну інструкцію у кількох рядках. Для цього її необхідно обмежити круглими, квадратними або фігурними дужками:

```
y = (x**5 + x**4 +
        x**3 + x**2 + x + 1)
```

- Вкладені інструкції об'єднуються в блоки за величиною відступів. Для відступу, як правило використовують 4 пробілу;
- Вкладені інструкції в Python записуються відповідно до одного і того ж шаблону, коли основна інструкція завершується двокрапкою, слідом за якою розташовується вкладений блок коду, зазвичай з відступом під рядком основної інструкції

```
Основна інструкція:
    Вкладений блок інструкцій
```

Ідентифікатор – це послідовність символів, що може використовуватися як ім’я елементу (об’єкту) у мові програмування.

У Python у ролі ідентифікаторів може використовуватися послідовність символів, що складається з літер латинського алфавіту (при цьому розрізняються великі та маленькі літери), цифр та знаку

нижнього підкреслення «_», при умові, що першим символом не є цифра.

Приклади ідентифікаторів у Python:

```
y
_error777
variable__111
```

Константа – спосіб адресації даних, зміна яких програмою не передбачається або забороняється.

Константи поділяються на два види – **літерали** і **іменовані константи. Літерал** – стале значення певного типу даних, записане у вихідному коді комп'ютерної програми.

```
256              # Числовий літерал цілого типу
0.1              # Числовий літерал дійсного типу
3.14159          # Числовий літерал для числа пі
"Рядок-літерал"  # Рядковий літерал
```

Як випливає з означення літерали розрізняються за типами. Типи літералів визначають не лише зображення літералу, але й операції, які можна проводити над цими літералами. Наприклад, для літералів числових типів визначені арифметичні операції, операції порівняння тощо.

Рядкові літерали беруть у апострофи, подвійні лапки, потрійні апострофи або потрійні подвійні лапки, для того, щоб відрізняти їх від ідентифікаторів.

```
'Рядок-літерал'

"Рядок-літерал"

'''Це рядок-літерал, який можна
   розміщувати у кількох рядках'''

"""Це також рядок-літерал, який можна
   розміщувати у кількох рядках"""
```

Іменована константа відрізняється від літералу тим, що крім сталого значення вона ще має ім'я. Таким чином, замість значення літералу можна використовувати це ім'я. Це зручно, наприклад, в тому разі, коли певний літерал використовується багаторазово або має велику довжину. Слід зауважити, що у Python немає окремої конструкції

для створення іменованих констант. Тому іменовані константи позначаються аналогічно до змінних.

```
pi = 3.14159      # pi - іменована константа для числа пі
```

Змінна – іменована область пам'яті, ім'я якої (ідентифікатор) використовується для здійснення доступу до даних (значення змінної), що містяться у цій області пам'яті.

Значення змінної є літералом. Тип змінної визначається типом літералу, що є значенням цієї змінної.

```
x = 23      # x - змінна зі значенням 23
y = x + 3   # y - змінна зі значенням 26
```

При роботі зі змінними у Python потрібно пам'ятати, що змінна є посиланням на область пам'яті, де зберігаються її дані. Наприклад, якщо

```
a = 3
s = a
```

то це фактично означає, що використовується не дві різних змінних `a` та `s`, а одна змінна зі значенням 3, що має два різних імені `a` та `s`.

Змінні у Python поділяються на два види:

- **змінювані** (**mutable**)
- **незмінювані** (**immutable**).

До об'єктів змінюваних типів є безпосередній доступ для модифікації даних цих об'єктів (без створення нових). Модифікація даних об'єктів незмінюваних типів можлива лише через створення нових об'єктів (тобто через виділення пам'яті).

Змінні всіх простих типів (`int`, `float`, `complex`) належать до незмінюваних. При вивченні нових типів даних будемо зауважувати, до якого виду відноситься кожен з них.

Ключові слова мови програмування – це зарезервовані ідентифікатори, що наділені певним сенсом. Їх можна використовувати виключно відповідно до значення яке закріплене за ними у мові програмування.

Зауваження. Оголошуючи новий ідентифікатор у якості імені змінної, функції або іншого об'єкту, потрібно переконатися, що цей ідентифікатор не є ключовим словом мови програмування. Інакше програма або не виконається, або виконається з помилками.

Сучасні інтегровані середовища програмування, як правило ключові слова виділяють іншим кольором. Тому пишучи програму, ви скоріше за все зрозумієте, що цей ідентифікатор не можна використовувати у ваших цілях. Наприклад, у коді нижче ідентифікатори **while** і **if** є ключовими словами.

```
while i <= N - 1:
    if N % i == 0:
        prime = False
    i = i + 1
```

Коментування коду

Коментарі – зрозуміла для програміста анотація, що знаходиться безпосередньо у вихідному коді комп'ютерної програми.

Коментарі пишуть для того щоб зробити код більш зрозумілим. Вони ігноруються при компіляції й інтерпретації. Коментарі також обробляють різними способами для створення окремої документації на базі вихідного коду за допомогою генераторів документації.

Коментарі поділяють на **однорядкові** та **багаторядкові**.

Однорядковий коментар може розташовуватися лише в одному фізичному рядку програми. Починається символом # і закінчується кінцем рядка.

```
x = 23      # Цей текст коментар
```

Багаторядковий коментар може розташовуватися у кількох фізичних рядках програми. Виділяється з двох боків потрійними апострофами або потрійними подвійними лапками

```
'''Багаторядковий коментар може
розташовуватися у кількох
фізичних рядках програми'''
```

Основні найпростіші команди

Тотожна інструкція

Тотожна інструкція задається командою

```
pass
```

Інтерпретатор не виконує жодних дій, коли зустрічає цю інструкцію та переходить до виконання наступної інструкції. Основне її призначення – використовуватися там, де інструкція вимагається синтаксисом мови, проте ніяких дій виконувати не потрібно.

Присвоєння

Присвоєння — одна з центральних конструкцій в імперативних мовах програмування.

Присвоєння – механізм у програмуванні, що дозволяє динамічно змінювати зв'язки об'єктів даних (наприклад, змінних) із їхніми значеннями.

Синтаксис присвоєння у Python такий

```
x = e
```

тут `e` – змінна (об'єкт) або арифметичний вираз (інструкція, що повертає результат), який може бути записаний у змінну. `x` – змінна, у яку записується результат `e`.

Після інструкції присвоєння попереднє значення змінної `x` стає недосяжним.

```
x = 12
y = (x**5 + x**4 + x**3 + x**2 + x + 1)
```

Тут змінній `x` присвоюється значення `12`, а змінній `y` значення арифметичного виразу, у якому бере участь змінна `x`.

Введення та виведення

Інструкції введення та виведення використовуються для взаємодії користувача та програми, а також для обміну інформацією між зовнішніми носіями та оперативною пам’яттю комп’ютера (об’єктами програми). Існують різноманітні способи введення і виведення

інформації з використанням клавіатури, екрану, файлів, звукової карти комп'ютера, принтера та інших периферійних пристроїв. Проте, наразі під введенням будемо розуміти введення користувачем певної інформації з клавіатури під час виконання програми, а під виведенням – друкуванням даних у консоль під час виконання програми.

Інструкція введення має такий синтаксис

```
x = input(S)
```

де `x` – змінна, у яку записується результат введення з клавіатури, `S` – рядок підказки.

Коли інтерпретатор зустрічає інструкцію введення, він призупиняє роботу програми і очікує, коли з клавіатури буде щось введено. При цьому на екран виводиться рядок підказки `S`. Результат введення записується у змінну, що стоїть зліва від знаку рівності.

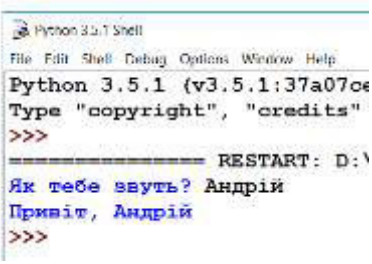
Синтаксис інструкції виведення такий

```
print(S1,…,SN)
```

де `S1,…,SN` – вирази, що виводяться на екран. Виведення виразів відбувається через символ пропуску.

Для прикладу розглянемо таку просту програму

```
name = input("Як тебе звуть? ")
print("Привіт,", name)
```



Результат виконання цієї програми зображено на скріншоті. Як бачимо програма вивела повідомлення "Як тебе звуть? ", та дочекавшись введення з клавіатури, вивела повідомлення "Привіт, Андрій".

Інструкція виведення `print` має два необов'язкових параметри `sep` та `end`. Вони вказують які символи використовуються у якості розділювача та кінця рядка відповідно. Типовими значеннями є відповідно символ пропуску та кінець рядка. Для прикладу проаналізуйте результат виконання коду.

```
>>> print(1,2,3)
1 2 3
>>> print(1,2,3, sep="_")
1_2_3
```

```
>>> print(1); print(2)
1
2
>>> print(1, end="!"); print(2, end="!")
1!2!
```

Досить часто виникає ситуація, коли необхідно вивести рядок (повідомлення), згідно з заданим шаблоном, підставивши у визначені позиції дані, отримані у результаті виконання програми. Таку підстановку можна здійснювати за допомогою оператора `%`.

Оператор `%` підставляє значення виразів, що стоять справа від нього у визначені спеціальними символами позиції. Спеціальні символи це послідовності з двох символів, перший з яких символ "`%`". Таблиця спеціальних символів наведена нижче

Таблиця 1.1. Найуживаніші спеціальні символи

Формат	Тип

`%d`
`%i`
`%u`
	Десяткове число ціле число.

`%e` `%E`	Дійсне число в експоненційній формі.
`%f` `%F`	Дійсне число у звичайному форматі.
`%c`	Символ (або код символа).
`%s`	Рядок символів.
`%%`	Символ %.

Розглянемо на прикладах

```
>>> "First = %d, second = %d" % (1, 2)
First = 1, second = 2
```

Справа від оператора `%` стоїть у дужках послідовність з двох чисел 1 та 2. Як бачимо, інтерпретатор, замість першого спеціального символа "`%d`" вставив цифру 1, а замість другого – 2. Кількість підстановок за допомогою оператора `%` може бути довільна. Єдине правило, якого потрібно дотримуватися – кількість спеціальних символів у рядку зліва від оператора `%` має дорівнювати кількості значень справа від оператора `%`.

```
>>> print("%s було %d років, коли він пішов у %d клас" %
("Миколі", 7, 1))
```

```
Миколі було 7 років, коли він пішов у 1 клас
```

Якщо у операторі підстановці використовується лише одне значення дужки справа від оператора `%` можна опустити

```
>>> print("Високосний рік має %d днів" % 366)
Високосний рік має 366 днів
```

Оператор `%` можна використовувати скрізь де необхідно зробити підстановку, зокрема у його можна використовувати у інструкції введення `input`, наприклад,

```
x = input("Введіть %d-й член послідовності " % 10)
```

Лінійна програма

Лінійна програма – це програма, що складається лише з інструкцій введення, виведення, присвоєння та тотожної команди.

Числові типи даних

Числа у Python нічим не відрізняються від звичайних чисел та підтримують математичні операції відомі вам з математики. Числові типи у Python представлені трьома типами: цілими, дійсними та комплексними. Розглянемо детальніше кожен з них.

Цілі числа

Змінні та літерали цього типу набувають значень з (обмеженої) множини цілих чисел. Цілий тип у Python, при необхідності, позначають `int`.

Таблиця 1.2. Арифметичні операції для цілого типу

Операція	Опис
`x + y`	сума x та y
`x - y`	різниця x та y
`x * y`	добуток x та y
`x / y`	частка від ділення x на y
`x // y`	ділення націло x на y
`x % y`	остача від ділення x на y
`-x`	унарний мінус – повертає x протилежного знаку
`x ** y`	x піднесене до степеня y

```
>>> x = 1 + 2
>>> print(x);
3
```

```
>>> x = 11 // 2
>>> print(x);
5
```

```
>>> x = 2 ** 4
>>> print(x);
16
```

Таблиця 1.3. Базові функції для цілого типу

Функція	Опис
`abs(x)`	повертає модуль x
`int(x)`	результатом буде перетворення x до цілого
`float(x)`	результатом буде перетворення x до дійсного
`divmod(x, y)`	пара (x // y, x % y)

```
>>> x = int(9.6)
>>> x
9
```

```
>>> x = abs(-52)
>>> print (x)
52
```

Таблиця 1.4. Пріоритет операцій від найвищого до найнижчого

`**`
`*, /, //, %`
`+, -`

Пріоритет унарного мінуса залежить від того, з якою операцією він використовується.

Для прикладу розглянемо такий код

```
>>> -3 ** 2
-9
>>> 3 ** -2
0.1111111111111111
```

Як бачимо у першому випадку початку виконується операція піднесення до степеня, а вже потім застосовується унарний мінус, а в другому випадку навпаки.

Пріоритет арифметичних операцій, можна змінювати за допомогою дужок:

```
>>> 2 + 2 * 2
6
>>> (2 + 2) * 2
8
```

Приклад 1.1. Знайти суму цифр двозначного числа.

Розв'язок. Нехай x – задане двозначне число, наприклад 25. Сума цифр цього числа є сумою першої та другої цифр цього числа: $2 + 5 = 7$. Першу цифру цього числа можна знайти як ділення числа x націло на 10. Друга ж цифра цього числа є остачею від ділення числа x на 10. Тоді програма буде мати вигляд

```
x = int(input("Введіть двозначне число "))
first = x // 10  # first - перша цифра числа
second = x % 10  # second - друга цифра числа
suma = first + second
print("Сума цифр числа %d = %d" % (x, suma))
```

Зауважимо, що операція input повертає літерал рядкового типу. Тому у першому рядку програми, відбувається перетворення результату введення з клавіатури до цілого числа за допомогою інструкції int.

Результатом виконання цієї програми для введеного числа 25 буде

```
Введіть двозначне число 25
Сума цифр числа 25 = 7
```

Дійсні числа

Дійсні числа у Python записуються у вигляді десяткового дробу, для розділення цілої і дробової частини у якому застосовується символ крапка. Дійсний тип у Python, позначають `float`

```
>>> 3.11
3.11
```

Щоб відрізняти літерали дійсних чисел без дробової частини від цілих, записують десятковий дріб з нульовою дійсною частиною

```
>>> 3.0
3.0
>>> type(3)
<class 'int'>
>>> type(3.0)
<class 'float'>
```

Функція `type` повертає тип літералу. Як бачимо в першому випадку цілий тип, а в другому – дійсний.

Крім зображення дійсних чисел у вигляді звичайного десяткового дробу використовується експоненціальних або показниковий запис.

Експоненціальний запис (формат) – зображення дійсних (дробових) чисел у вигляді мантиси і порядку:

$$N = M \cdot n^p,$$

де N – число, M– мантиса, n – основа показникової функції, p – порядок.

В інформатиці основу показникової функції n беруть рівною 10, а комп'ютерну реалізацію чисел у експоненціальній формі називають *числами з плаваючою комою* (*плаваючою крапкою*). Показник степеня в обчислювальних машинах прийнято відділяти від мантиси символами `"E"` або `"e"` (скорочення від "exponent"). Наприклад, число

$$1.234 \cdot 10^{-56}$$

записується так:

```
1.1234e-56
```

Слід зауважити, що зображення дійсного числа в показниковій формі є неоднозначним. Наприклад, число 0.0001 у показниковій формі можна записати багатьма способами, наприклад,

$$0.0001 = 10 \cdot 10^{-5} = 0.1 \cdot 10^{-3}.$$

Тому в інформатиці використовують нормалізовану форму запису чисел з плаваючою комою.

Нормалізованою формою запису числа плаваючою комою називають зображення, у якому мантиса належить проміжку $[1,10)$.

У такій формі будь-яке число (крім нуля) записується єдиним чином. Наприклад, вищенаведене число 0.0001 у нормалізованій формі буде мати вигляд:

$$0.0001 = 1 \cdot 10^{-4}.$$

Над змінними та літералами дійсного типу можна проводити ті ж арифметичні операції, що і для цілого типу наведені у таблицях 1.2 та 1.3. Пріоритет арифметичних операцій також зберігається згідно з таблицею 1.4.

```
>>> 3.0 * 2.0
6.0
```

```
>>> 9.0 ** 0.5
3.0
```

Слід зауважити, що через двійкове кодування дійсних чисел у пам'яті комп'ютера, дійсні числа не є точними, що може призводити до неочікуваних, на перший погляд користувача, результатів.

Розглянемо на прикладі.

```
>>> res1 = 0.5+0.5
>>> res2 = 0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1
```

З точки зору математики змінні `res1` та `res2`, мають містити однаковий результат. Проте, вивівши їхні значення на екран

```
>>> res1
1.0
>>> res2
0.9999999999999999
```

переконуємося, що це не так.

Оперуючи дійсними числами завжди потрібно враховувати похибку арифметичних операцій.

При роботі з числовими типами даних, Python, без додаткових застережень (якщо це допустимо математичними правилами), допускає використання у арифметичних операціях операндів різних числових типів, наприклад, дійсного і цілого. У такому арифметичному виразі всі

операнди (або результат при необхідності) будуть автоматично зведені до типу з найширшими можливостями серед представлених.

```
>>> 4.5 + 4
8.5
>>> 9 ** -1
0.1111111111111111
```

Приклад 1.2. Обчислити значення многочлена

$$y = x^6 - 4x^4 + 3x - 7$$

для заданого значення x.

Розв'язок. Програма буде мати вигляд

```
x = float(input('x=? '))
y = x ** 6 - 4 * x ** 4 + 3 * x - 7
print('y=', y)
```

Окрім основних арифметичних операцій та функцій наведених вище, Python надає програмісту математичну бібліотеку `math`, яка містить додаткові математичні функції і сталі. Для використання бібліотеки її необхідно імпортувати командою

```
import math
```

Команди імпортування бібліотек, як правило, записують на початку файлу, що містить вихідний код. Проте можна проводити імпорт бібліотеки безпосередньо перед використанням функцій з неї.

Таблиця 1.5. Функції та сталі з бібліотеки math

Функція	Опис
`math.pi`	Константа $\pi = 3.141592...$
`math.e`	Константа $e = 2.718281...$
`math.sqrt(x)`	Обчислення функції $\sqrt{x}$
`math.exp(x)`	Обчислення функції e^x
`math.log(x)`	Обчислення функції $\ln x$
`math.log(x, a)`	Обчислення функції $\log_a x$
`math.log1p(x)`	Обчислення функції $\ln(1+x)$
`math.log2(x)`	Обчислення функції $\log_2 x$
`math.log10(x)`	Обчислення функції $\lg x$
`math.cos(x)`	Обчислення функції $\cos x$
`math.sin(x)`	Обчислення функції $\sin x$

Функція	Опис
`math.tan(x)`	Обчислення функції $\mathrm{tg}\, x$
`math.acos(x)`	Обчислення функції $\arccos x$
`math.asin(x)`	Обчислення функції $\arcsin x$
`math.atan(x)`	Обчислення функції $\mathrm{arctg}\, x$
`math.atan2(y, x)`	Обчислення функції $\mathrm{arctg}\frac{y}{x}$
`math.cosh(x)`	Обчислення функції $\mathrm{ch}\, x$
`math.sinh(x)`	Обчислення функції $\mathrm{sh}\, x$
`math.tanh(x)`	Обчислення функції $\mathrm{th}\, x$
`math.hypot(x, y)`	Обчислення $\sqrt{x^2+y^2}$
`math.degrees(x)`	Повертає результат перетворення x з радіан у градуси
`math.radians(x)`	Повертає результат перетворення x з градусів у радіани
`math.factorial(n)`	Обчислення n! для цілого та невід'ємного значення аргументу.
`math.trunc(x)`	Дійсне число, яке є результатом відкидання дробової частини x
`math.ceil(x)`	Найменше ціле, яке більше або дорівнює x
`math.floor(x)`	Найбільше ціле, яке менше або дорівнює x

Приклад 1.3. Знайти кути заданого прямокутного трикутника.

Розв'язок. Будемо вважати, що прямокутний трикутник задається двома катетами a та b. Тоді, один з його гострих кутів визначається як

$$\beta = arctan\frac{a}{b}.$$

Отже, програма буде мати вигляд

```
import math              # імпорт математичної бібліотеки

a = float(input("Введіть перший катет "))
b = float(input("Введіть другий катет "))
beta = math.atan2(a, b)      # один з кутів у радіанах
betaDegree = math.degrees(beta) # перетворення у градуси

print("Перший кут = ", betaDegree)
print("Другий кут = ", 90.0 - betaDegree)
```

Результатом виконання цієї програми буде (наприклад, для двох однакових катетів одиничної довжини):

```
Введіть перший катет 1
```

```
Введіть другий катет 1
Перший кут =  45.0
Другий кут =  45.0
```

Комплексні числа

Комплексні числа у Python задають парою – дійсною і уявною частиною. Для позначення уявної частини, поруч з її значенням використовують символ `j`. Комплексний тип у Python, позначають `complex`

```
>>> c = 2 + 3j
>>> print(c)
(2+3j)
```

```
>>> c = complex(2,3)
>>> print(c)
(2+3j)
```

Вводити з клавіатури комплексні числа треба також у алгебраїчній формі використовуючи у якості уявної одиниці символ `j`

```
>>> c = complex(input("z = "))
z = 2+3j
>>> print(c)
(2+3j)
```

Для комплексних чисел у Python визначені чотири основні арифметичні операції, а також операція унарний мінус і піднесення до степеня.

Таблиця 1.6. Арифметичні операції для комплексного типу

Операція	Опис
`x + y`	сума x та y
`x - y`	різниця x та y
`x * y`	добуток x та y
`x / y`	частка від ділення x на y
`-x`	унарний мінус – повертає x протилежного знаку
`x ** y`	x піднесене до степеня y

Крім цього для комплексних чисел визначені такі базові функції

Таблиця 1.7. Базові функції для комплексного типу

Функція	Опис
`abs(z)`	модуль комплексного числа z
`complex(re, im)`	створення комплексного числа з пари дійсних чисел re та im.
`z.real`	дійсна частина числа z
`z.imag`	уявна частина числа z
`z.conjugate()`	комплексно-спряжене число до z

Розширений набір функцій для роботи з комплексними числами містить бібліотека `cmath`. Вона містить додаткові функції і сталі наведені у таблиці.

Таблиця 1.8. Функції та сталі з бібліотеки cmath

Функція	Опис
`cmath.pi`	Константа $\pi = 3.141592...$
`cmath.e`	Константа $e = 2.718281...$
`cmath.sqrt(z)`	Обчислення функції $\sqrt{x}$
`cmath.exp(z)`	Обчислення функції e^z
`cmath.log(z)`	Обчислення функції $\ln z$
`cmath.log(z, a)`	Обчислення функції $\log_a z$
`cmath.log10(z)`	Обчислення функції $\lg z$
`cmath.cos(z)`	Обчислення функції $\cos z$
`cmath.sin(z)`	Обчислення функції $\sin z$
`cmath.tan(z)`	Обчислення функції $\mathrm{tg}\, z$
`cmath.acos(z)`	Обчислення функції $\arccos z$
`cmath.asin(z)`	Обчислення функції $\arcsin z$
`cmath.atan(z)`	Обчислення функції $\mathrm{arctg}\, z$
`cmath.cosh(z)`	Обчислення функції $\mathrm{ch}\, z$
`cmath.sinh(z)`	Обчислення функції $\mathrm{sh}\, z$
`cmath.tanh(z)`	Обчислення функції $\mathrm{th}\, z$
`cmath.phase(z)`	Аргумент комплексного числа `z`
`cmath.polar(z)`	Зображення комплексного числа z у полярних координатах.
`cmath.rect(r,phi)`	Перетворення комплексного числа з полярними координатами `r`, `phi` у алгебраїчну форму.

Приклад 1.4. Доведемо зв'язок між трьома магічними сталими e, π, i:

$$e^{i\pi} = -1$$

Розв'язок. Скористаємося бібліотекою `cmath`

```
import cmath              # підключаємо модуль cmath

i = 1j                    # задамо уявну одиницю
pi = cmath.pi             # задамо число пі
z = cmath.exp(i * pi)
print(z)
```

Результатом виконання вищенаведеної програми буде виведення на екран числа

```
(-1+1.2246467991473532e-16j)
```

У якому уявна частина майже дорівнює нулю.

Задачі для аудиторної роботи

1.1. Вивести на екран таблицю

```
 x | 1 | 2 | 3 | 4 | 5
---+---+---+---+---+---
 y | 3 | 1 | 5 | 4 | 2
```

1.2. Дано натуральне тризначне число. Знайти суму цифр цього числа.

1.3. Обчислити гіпотенузу с прямокутного трикутника за катетами a та b.

1.4. Наближено визначити значення числа π, використовуючи ланцюговий дріб

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292}}}}.$$

1.5. Обчислити значення многочлена для введеного з клавіатури комплексного числа z:

a) $f(z) = 4z^4 + 3z^3 + 2z^2 + z + 1$; b) $f(z) = z^4 + 2z^2 + 1$;